

Trabalho Prático 2 -

Implementação de Protocolos de Aplicação para

Missões Espaciais: MissionLink (Via UDP) e

TelemetryStream (Via TCP)

Comunicações por Computador

Hugo Rauber
A104534

Rui Fernandes
A76231

Tomás Machado
A104186



Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática
Unidade Curricular de Comunicações por Computador
Outubro 2025

Índice

Introdução	4
Arquitetura e Topologia	5
Nós da Rede	6
Pacote de mensagens	7
Protocolo MissionLink	9
Arquitetura e Protocolo de Transporte	9
Definição de Missão	9
Mecanismos de Fiabilidade	9
Fragmentação e Suporte a Grandes Volumes	10
Diagrama de Sequência	10
Protocolo TelemetryStream	11
Arquitetura e Protocolo de Transporte	11
Especificação dos Dados	11
Controlo de Fluxo e Delimitação	11
Mecanismos de Fiabilidade e Recuperação	12
Diagrama de Sequência	12
Nave-Mãe em HTML	13
Arquitetura de Concorrência	13
Servidor e Integração Web	13
Gestão de Estado Partilhado	14
Ground Control em HTML	15
Funcionalidades de Monitorização	15
Implementação da API de observação	16
Testes Realizados	17
Cenário e Configuração de Testes	17
Análise dos Resultados	17
Conclusão e reflexão crítica	20

Listas de Figuras

Figura 1 Arquitetura de Concorrência da Nave-Mãe	5
Figura 2 Topologia usada	6
Figura 3 Diagrama de Sequência do Protocolo MissionLink	10
Figura 4 Diagrama de Sequência do Protocolo TelemetryStream	12
Figura 5 Dashboard da Nave-Mãe	13
Figura 6 DashBoard do GroundControl	15
Figura 7 Secção do Gound control onde se pode ver o histórico da missão do Rover-Alpha	15
Figura 8 Tipologia para os testes realizados	17
Figura 9 Teste do Rover-Alpha	17
Figura 10 Teste do Rover-Beta	18
Figura 11 Teste do Rover-Gamma	18

Lista de Tabelas

Tabela 1	Pacote de mensagens detalhado	7
Tabela 2	Tabela de endpoints implementados	16

Introdução

A comunicação de dados em ambientes hostis e remotos representa um dos maiores desafios da engenharia moderna, sendo crítica para o sucesso de operações complexas como a exploração espacial. Neste contexto, a fiabilidade e a eficiência dos protocolos de rede não são apenas requisitos desejáveis, mas sim condições imperativas para garantir a integridade da missão e a segurança dos equipamentos.

O presente trabalho prático, desenvolvido no âmbito da Unidade Curricular de Comunicações por Computador, tem como objetivo principal o desenho, implementação e validação de uma infraestrutura de comunicação completa para uma missão espacial simulada. O cenário proposto envolve a coordenação de uma frota de Rovers autónomos por uma Nave-Mãe em órbita, num ambiente sujeito a elevada latência, perda de pacotes e instabilidade de rede.

Para responder a estes desafios, o projeto focou-se no desenvolvimento de dois protocolos aplicacionais distintos sobre a pilha TCP/IP:

MissionLink (sobre UDP): Um protocolo desenhado para o envio de comandos críticos e transferência de ficheiros, onde foi necessário implementar mecanismos de fiabilidade na camada de aplicação para compensar a natureza “best-effort” do transporte UDP.

TelemetryStream (sobre TCP): Um protocolo orientado a fluxo para a monitorização contínua do estado dos rovers (telemetria), garantindo a entrega ordenada e íntegra de dados vitais como níveis de bateria e posicionamento geográfico.

Adicionalmente, para permitir a supervisão em tempo real da missão, foi desenvolvida uma **API** de Observação (REST) e uma interface da **Nave-mãe**, para gerir as missões e **Ground Control**, integrando os dados da rede num dashboard visual ambas acessíveis via Web.

Todo o sistema foi validado utilizando o emulador de redes **CORE**, permitindo testar a robustez da solução sob condições adversas controladas, como taxas de perda de pacotes (BER) e duplicação, simulando com realismo as dificuldades de um canal de comunicação espacial.

Este relatório detalha a arquitetura da solução, as decisões de desenho tomadas e apresenta uma análise crítica dos resultados obtidos nos testes de carga e falha.

Arquitetura e Topologia

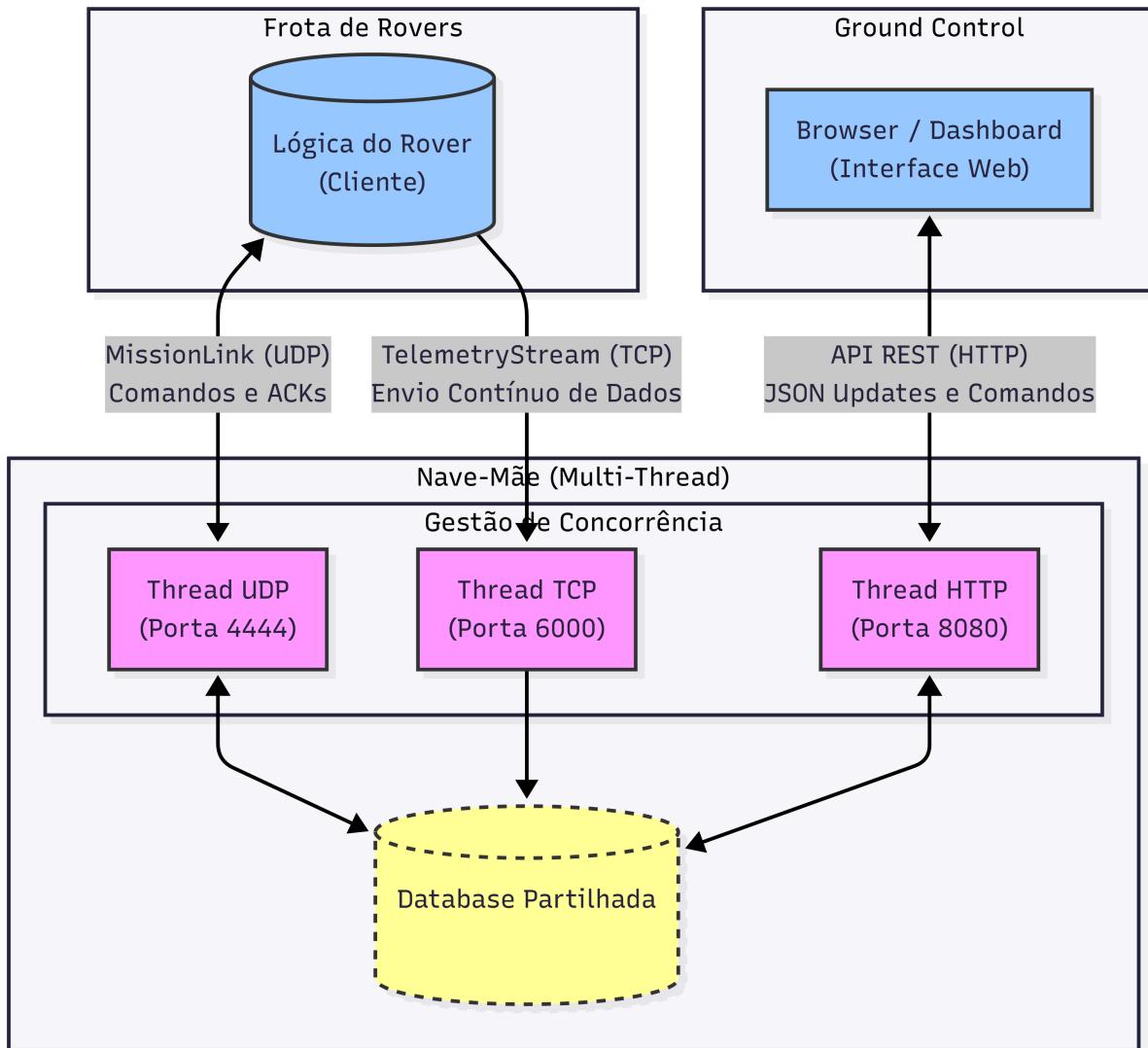


Figura 1: Arquitetura de Concorrência da Nave-Mãe

O diagrama ilustra o modelo *multithreaded* do servidor, onde três *threads* independentes gerem os protocolos de comunicação (UDP, TCP, HTTP) em paralelo, partilhando o estado global através de uma estrutura de dados protegida (Thread-Safe).

Conforme ilustrado na Figura 1, a arquitetura interna da Nave-Mãe foi desenhada para desacoplar a gestão da rede do processamento lógico. O sistema opera com três *threads* principais que correm simultaneamente, evitando que operações de bloqueio (como a espera por um pacote UDP) congelem a interface de utilizador ou a receção de telemetria:

Thread MissionLink (UDP - Porta 4444): Responsável pela troca de mensagens críticas (Comandos e ACKs). Esta thread lê e escreve na Base de Dados partilhada para atualizar o estado das missões.

Thread TelemetryStream (TCP - Porta 6000): Gere conexões persistentes com a frota. Os dados recebidos (bateria, posição) fluem num sentido único para a Base de Dados, garantindo que a informação mais recente está sempre disponível.

Thread API REST (HTTP - Porta 8080): Atua como interface para o Ground Control. Sempre que o operador solicita uma atualização visual, esta thread consulta a Base de Dados em tempo real e devolve o estado da frota em formato JSON.

O sistema foi emulado utilizando a ferramenta **CORE**, com uma topologia estática definida no ficheiro **TP2_IMM**.

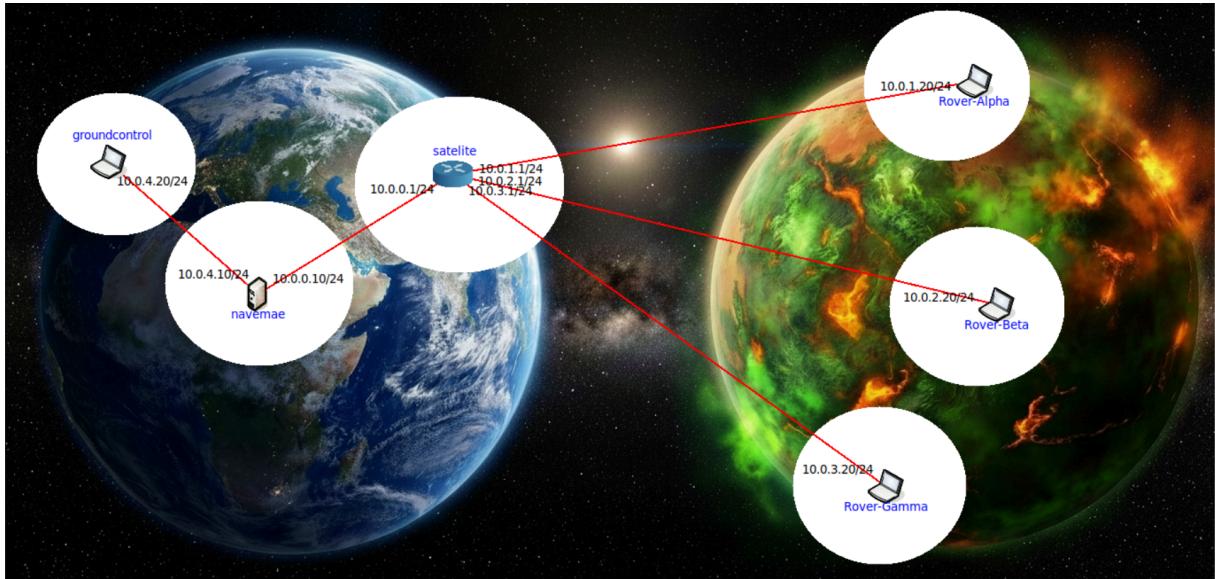


Figura 2: Topologia usada

Para a topologia o grupo de trabalho decidiu colocar estes **6 elementos** na ferramenta **Core**:

- 4 PCs referentes ao groundcontrol, e os rovers Alpha, Beta e Gamma com os respetivos IP's:
 - 10.0.4.20, 10.0.1.20, 10.0.2.20 e 10.0.3.20;
- 1 router que será o satélite que servirá como ponto multisalto entre os rovers e a Nave-Mãe;
- 1 Host que será a Nave-Mãe visto que esta recebe e envia todas as informações;

Nós da Rede

- Nave-Mãe (10.0.0.10): Servidor central que agrupa telemetria e gera as missões.
- Satélite (Router): Atua como ponto intermediário de comutação.
- Rovers (Alpha, Beta, Gamma): Clientes autónomos situados na sub-rede 10.0.x.20.
- Ground Control (10.0.4.20): Cliente de visualização ligado à Nave-Mãe.

Pacote de mensagens

Foi desenvolvido um pacote de mensagens específico. A tabela seguinte detalha os campos contidos nestas mensagens, bem como o espaço (em bytes) que cada um ocupa:

	Tipo_MSG	Num_Seq	ACK	Flags	Frag-ment Offset	Tama-nho do Payload	Dados do Pay-load
Sem Frag-menta-ção	1	2	2	1	2	2	0
Com Frag-menta-ção	1	2	2	1	2	2	1400 *

Tabela 1: Pacote de mensagens detalhado

*Máximo de 1400 Bytes

Onde cada campo possuí os seguintes significados:

TIPO MSG (1 Byte): Identifica o propósito do pacote.

- 1: Pedido de Missão
- 2: Dados da Missão (Comandos, Fotos)
- 3: ACK (Confirmação)
- 4: Progresso (Status, Bateria)

NUM_SEQ (2 Bytes): O número de identidade deste pacote (ex: 101, 102). Usado para ordenar e detetar duplicados.

ACK_NUM (2 Bytes): Se o pacote for do tipo ACK (3), este campo diz qual o número de sequência que está a confirmar (ex: “Recebi o 101”).

FLAGS (1 Byte): Controla a fragmentação.

- 1: MORE_FRAGS (Ainda há mais pedaços da foto a caminho).
- 0: Este é o último pedaço (ou mensagem única).

FRAGMENT OFFSET (2 Bytes): Diz onde é que este pedaço encaixa no ficheiro original.

- Exemplo: Se o pacote leva 250 bytes, o primeiro tem Offset 0, o segundo Offset 250, o terceiro Offset 500.

TAMANHO PAYLOAD (2 Bytes): Indica quantos bytes de dados úteis vêm a seguir. Calculamos isto automaticamente (`len(self.payload)`).

PAYLOAD (Variável): A mensagem real (texto JSON ou bytes da foto).

É importante notar que a estrutura de pacote definida na Tabela 1 é exclusiva do protocolo MissionLink (sobre UDP). O protocolo TelemetryStream, por operar sobre TCP, não necessita de cabeçalhos de

controlo adicionais (como ACKs ou Sequência), consistindo apenas num fluxo contínuo de objetos JSON delimitados por quebras de linha.

Protocolo MissionLink

Arquitetura e Protocolo de Transporte

O **MissionLink** é o protocolo crítico responsável pela coordenação da missão, incluindo o envio de comandos da Nave-Mãe e a receção de confirmação e progresso por parte dos rovers.

De acordo com os requisitos, este protocolo foi implementado sobre UDP (User Datagram Protocol). Dado que o UDP é um protocolo “best-effort” (não garante entrega, ordem ou integridade), foi desenvolvida uma camada aplicacional robusta para assegurar a fiabilidade necessária para o controlo da missão.

Definição de Missão

Enquanto o cabeçalho (descrito na introdução) gera o transporte, o conteúdo da missão é encapsulado no payload em formato JSON. Esta escolha permite flexibilidade para incluir diversos parâmetros sem alterar a estrutura binária do protocolo.

Uma mensagem de atribuição de missão enviada pela Nave-Mãe contém os seguintes campos:

- **id**: identificador único da missão;
- **tarefa**: ação a realizar;
- **area**: Definição da zona alvo (coordenadas ou raio de ação);
- **duracao**: tempo máximo alocado para a execução (em segundos).
- **detalhes**: breve descrição da missão.
- **frequencia**: Define a periodicidade temporal (em segundos) com que o rover deve reportar o seu progresso à Nave-Mãe via UDP.

Exemplo de um payload de missão do sistema:

```
{  
    "id": "M-001",  
    "tarefa": "Coleta Mineral",  
    "area": [10.0, 20.0],  
    "duracao": 15,  
    "detalhes": "Missao de aquecimento.",  
    "frequencia": 2  
}
```

Mecanismos de Fiabilidade

Para compensar a falta de garantias do UDP, implementou-se um mecanismo de Stop-and-Wait ARQ (Automatic Repeat reQuest), garantindo retransmissão persistente de pacotes até confirmação (ACK) ou até atingir um limite de tentativas.

Este mecanismo estrutura-se da seguinte forma:

1. **Confirmação (ACKs)**: Sempre que um nó (Rover ou Nave) recebe um pacote válido (do tipo Dados ou Progresso), responde imediatamente com um pacote do tipo ACK contendo o mesmo número de sequência recebido.
2. **Retransmissão Automática**: O emissor mantém um temporizador. Se a confirmação (ACK) não for recebida dentro de uma janela de tempo (definida como 3 segundos), o pacote é retransmitido. O sistema realiza até 5 tentativas de envio. Caso não haja resposta após a 5^a tentativa, assume-se que o destino está incontactável.
3. **Deteção de Duplicados**: Devido às retransmissões, o receptor pode receber o mesmo pacote várias vezes (caso o ACK original se perca). Para evitar o processamento redundante, o protocolo mantém o registo do último número de sequência recebido. Pacotes com sequência inferior ou igual à atual

são descartados como duplicados, embora o ACK seja reenviado para garantir que o emissor cessa as tentativas de envio.

Fragmentação e Suporte a Grandes Volumes

Embora o UDP imponha limites ao tamanho do datagrama (MTU), o protocolo MissionLink implementa um mecanismo nativo de fragmentação para viabilizar a transferência de ficheiros volumosos. Esta funcionalidade é utilizada especificamente na transmissão de imagens.

O processo de envio ocorre da seguinte forma:

1. **Segmentação:** Quando uma imagem é capturada, esta é dividida em blocos sequenciais respeitando o limite máximo de carga útil (payload) configurado;
2. **Sinalização:** Cada pacote transporta informações cruciais para a reconstrução da imagem no cabeçalho, nomeadamente:
 - O campo “flags” ativa o bit de “Mais Fragmentos” (FLAG_MORE_FRAGMENTS) para indicar que existem mais fragmentos a caminho;
 - O campo “offset” sinaliza a posição exata do fragmento no ficheiro original.
3. **Reconstrução:** A Nave-mãe utiliza estes dados para concatenar os fragmentos na ordem correta, permitindo a reconstrução integral do ficheiro original.

Diagrama de Sequência

De seguida apresenta-se uma imagem ilustrativa do diagrama de sequência que representa a troca de mensagens através do protocolo **MissionLink**.

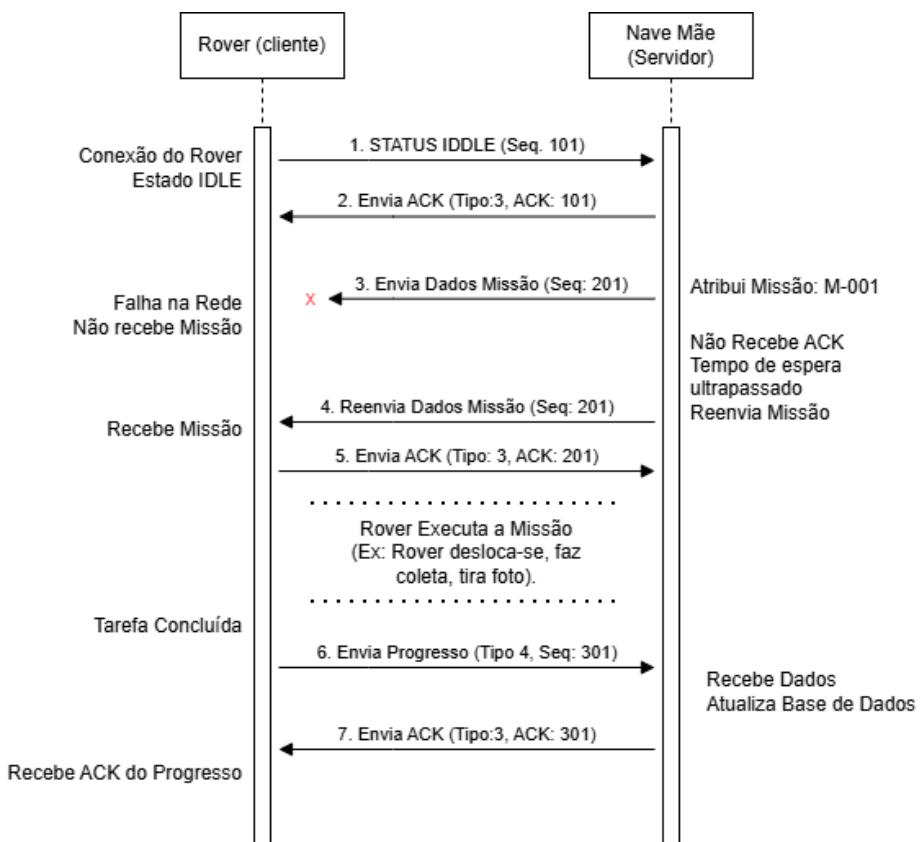


Figura 3: Diagrama de Sequência do Protocolo MissionLink

Protocolo TelemetryStream

Arquitetura e Protocolo de Transporte

O TelemetryStream (TS) é o protocolo responsável pela monitorização contínua e fiável do estado dos rovers. Conforme especificado no enunciado, o TS é implementado sobre o protocolo TCP (Transmission Control Protocol), garantindo um canal de comunicação persistente e fiável entre cada rover e a Nave-Mãe.

Embora a telemetria seja enviada de forma periódica, alguns campos — como o nível de bateria — requerem entrega fiável. Perdas ou receção fora de ordem poderiam originar leituras incoerentes devido a pacotes atrasados. A utilização de TCP assegura que todas as mensagens chegam pela ordem correta e sem perdas silenciosas, evitando inconsistências na base de dados central.

Assim, apesar do uso de TCP ser um requisito do projeto, o protocolo revela-se tecnicamente adequado às necessidades funcionais do sistema de telemetria.

Arquitetura Cliente-Servidor:

- **Servidor (Nave-Mãe):** Mantém a escuta de ligações TCP na porta 6000, preparado para receber múltiplos fluxos contínuos de telemetria em paralelo.
- **Cliente (Rovers):** Estabelece e mantém a conexão TCP, enviando periodicamente o estado atual.

Especificação dos Dados

Para a serialização dos dados, optou-se pelo formato JSON. O JSON permite uma estrutura legível, extensível e de fácil processamento.

Cada mensagem de telemetria representa o estado completo do rover num dado instante. Os campos e respetivos tipos são:

- **id:** int - identificador único do rover;
- **bat:** int - nível de bateria (0-100%);
- **pos:** [int, int, int] - posição no espaço (x,y,z);
- **status:** string - estado operacional atual.

Segue-se um exemplo de mensagem enviada do protocolo TelemetryStream:

```
{  
  "id": 2,  
  "bat": 73,  
  "pos": [214, 387, 0],  
  "status": "EM_MISSAO"  
}
```

Controlo de Fluxo e Delimitação

Uma vez que o protocolo TCP é orientado a fluxo (stream-oriented) e não a mensagens, não existe garantia de que uma leitura do socket corresponda exatamente a um objeto JSON. Os dados podem chegar fragmentados ou, inversamente, concatenados no mesmo segmento de rede (várias mensagens num só pacote).

Para mitigar este comportamento, implementou-se um mecanismo de buffering com separação por delimitadores. O algoritmo de reconstrução processa-se da seguinte forma:

1. **Acumulação:** os dados recebidos da rede são adicionados (concatenados) a uma variável string persistente;

2. **Delimitação:** o sistema procura ativamente pelo carácter de nova linha (\n), que atua como delimitador de fim de mensagem;
3. **Processamento:** Sempre que o delimitador é encontrado, o conteúdo acumulado até esse ponto é extraído e desserializado como um objeto JSON válido. Qualquer fragmento de dados restante é mantido no buffer para ser completado na próxima leitura.

Este mecanismo assegura a integridade dos dados, prevenindo erros de sintaxe JSON mesmo em cenários de fragmentação de rede severa.

Mecanismos de Fiabilidade e Recuperação

A robustez do sistema foi desenhada assumindo que a rede simulada pelo CORE é instável e propensa a falhas. Face a este cenário, foram implementados mecanismos de recuperação automática tanto no cliente como no servidor.

Cliente (Rover): Adota-se uma estratégia de reconexão persistente. Sempre que a comunicação é interrompida (por falha de rede ou timeout), o rover garante que a ligação anterior é fechada corretamente, e aguarda um intervalo de 5 segundos antes de tentar restabelecer contacto. Esta abordagem assegura que o veículo recupera a ligação à Nave-Mãe de forma totalmente autónoma, sem necessidade de intervenção humana.

Servidor (Nave-Mãe): O sistema foi desenhado para proteger o serviço central contra falhas individuais. O processamento de cada cliente é isolado, de modo que, se uma conexão gerar erros (por dados inválidos ou desconexão abrupta), o servidor deteta a falha e encerra apenas esse canal específico. Desta forma, evita-se que um problema num único rover afete a monitorização dos restantes veículos da frota.

Diagrama de Sequência

A figura seguinte ilustra a troca de mensagens entre um Rover e a Nave-Mãe através do protocolo TelemetryStream.

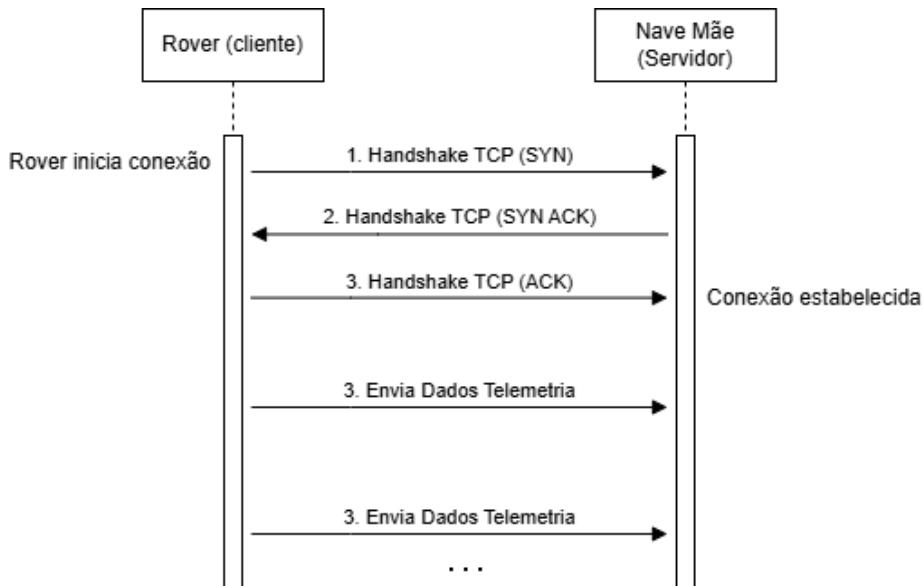


Figura 4: Diagrama de Sequ  ncia do Protocolo TelemetryStream

Nave-Mãe em HTML

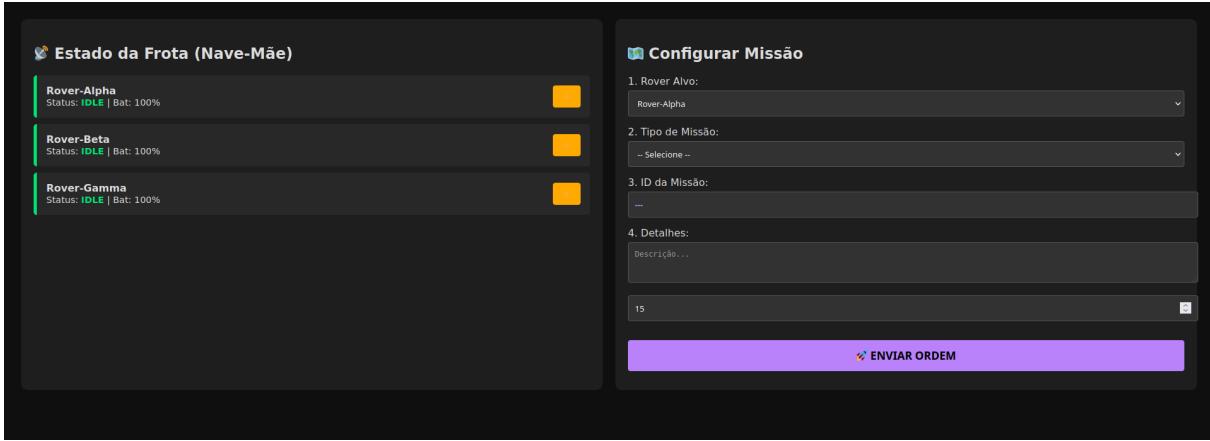


Figura 5: Dashboard da Nave-Mãe

A **Nave-Mãe** atua como o nó central da arquitetura, funcionando como um servidor multi-serviço que integra a persistência de dados, a comunicação com a frota e a interface de controlo. A sua implementação orquestra três componentes distintos que operam em paralelo: o **MissionLink** (UDP), o **TelemetryStream** (TCP) e a **API/Web Server** (HTTP).

Arquitetura de Concorrência

Devido à natureza assíncrona dos eventos (recepção de telemetria, envio de missões e pedidos do utilizador), a **Nave-Mãe** tem de adotar uma arquitetura baseada em multithreading.

A concorrência é gerida a dois níveis fundamentais:

1. **Nível de Serviço (Main Loop):** O processo principal não executa um ciclo de eventos único. Em vez disso, inicializa a base de dados partilhada e lança três **threads daemon** independentes:
 - **Thread UDP:** Responsável pelo protocolo MissionLink, mantendo a escuta na porta 4444.
 - **Thread TCP:** Dedicada ao TelemetryStream, aceitando conexões na porta 6000.
 - **Thread HTTP:** Serve a interface web e a API na porta 8080.

Esta separação garante que o processamento de um pedido HTTP ou a espera por um pacote UDP não bloqueia a receção de fluxos contínuos de telemetria via TCP.

2. **Nível de Conexão (Tratamento de Clientes):** Dentro dos serviços, a concorrência é refinada para garantir escalabilidade:
 - No **TelemetryStream**, cada nova conexão TCP aceite gera uma thread dedicada, permitindo que múltiplos rovers enviem dados em simultâneo sem contenção.
 - No **Servidor HTTP**, utiliza-se um servidor com suporte a **threading**. Isto assegura que cada pedido do browser (seja para carregar a interface ou atualizar dados) é tratado numa thread separada, mantendo o sistema responsivo.

Servidor e Integração Web

Para disponibilizar o componente **Ground Control**, a Nave-Mãe implementa um servidor HTTP que desempenha dois papéis simultâneos na porta 8080:

1. **Servidor de Aplicação (Frontend):** A **Nave-Mãe** entrega diretamente ao browser os ficheiros estáticos (HTML, CSS, JS) da interface. Isto elimina a necessidade de configurar servidores web externos, tornando a aplicação autocontida e fácil de executar.
2. **Gateway de Comandos (Backend):** A **API** expõe endpoints REST que interagem com os protocolos de baixo nível. Um exemplo crítico é o envio de missões: quando o utilizador ordena uma

tarefa na interface web, o pedido HTTP é intercetado pela Nave-Mãe, que serializa a missão em JSON e injeta-a diretamente no túnel UDP (MissionLink) em direção ao rover alvo.

Gestão de Estado Partilhado

A integridade dos dados num ambiente multithread é assegurada pela classe central `Database`. O acesso concorrente às estruturas de dados partilhadas é protegido por mecanismos de exclusão mútua (`threading.Lock`), aplicados em todos os métodos críticos de leitura e escrita (como `atualizar_telemetria` ou `get_novo_id_missao`).

Adicionalmente, o sistema implementa persistência de configuração: a lista de rovers é carregada do ficheiro “`rovers_config.json`” e o catálogo de missões de “`missões.json`”. Esta abordagem garante que a definição da frota e das tarefas se mantém consistente entre diferentes inicializações do sistema, sem necessidade de reconfiguração manual.

Ground Control em HTML

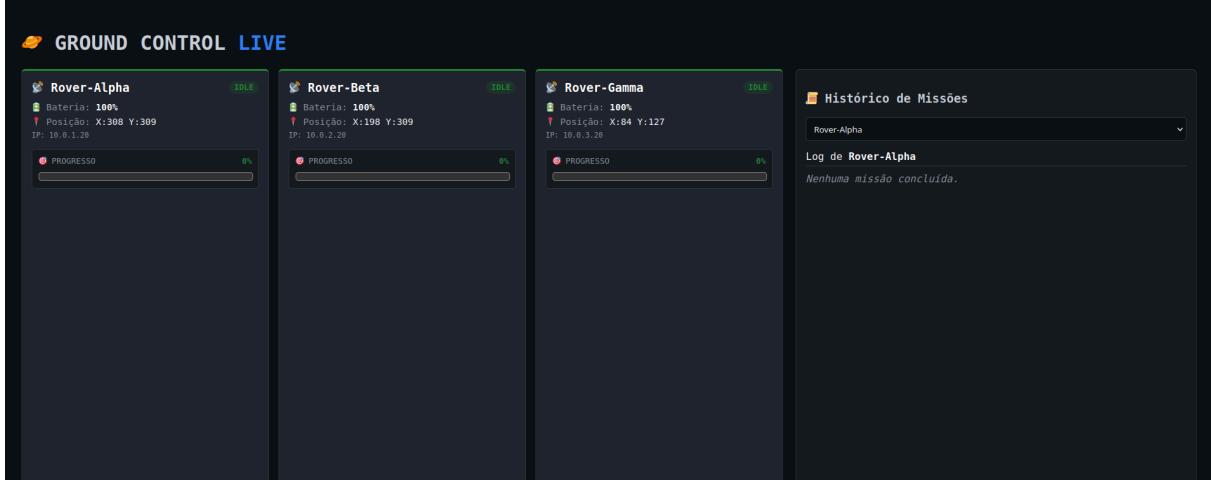


Figura 6: DashBoard do GroundControl

O **Ground Control** concretiza a interface de operação do sistema, permitindo acompanhar a missão.

Este componente foca-se na visualização de dados. A sua implementação tira partido do servidor HTTP embutido na Nave-Mãe para oferecer uma experiência de utilização simples e acessível a partir de qualquer ponto da rede.

Funcionalidades de Monitorização

A interface foi desenhada para apresentar de forma legível os dados agregados pela Nave-Mãe:

- Estado da Frota: uma lista dinâmica mostra todos os rovers ativos, onde é possível visualizar os dados de telemetria em tempo real para cada um.
- Visualização de Logs e histórico: um painel lateral exibe o histórico de eventos e o progresso das missões, permitindo confirmar se as tarefas foram concluídas com sucesso.



Figura 7: Secção do Gound control onde se pode ver o histórico da missão do Rover-Alpha

Implementação da API de observação

Para viabilizar a integração entre a **Nave-Mãe** e o **Ground Control**, bem como permitir a expansão para futuros clientes de monitorização, foi desenvolvida uma **API** que expõe o estado do sistema via HTTP na porta 8080.

A arquitetura do servidor baseia-se na classe ThreadingHTTPServer da biblioteca padrão do Python. Esta escolha é crucial para o desempenho do sistema, pois permite que o servidor processe múltiplos pedidos HTTP em threads separadas, sem bloquear o ciclo principal de processamento da Nave-Mãe ou a receção de telemetria via TCP/UDP.

A API responde exclusivamente em formato JSON, facilitando o parsing pelo cliente web.

A tabela abaixo detalha os endpoints implementados no módulo `src/services/api.py`:

Método	Endpoint	Descrição	Exemplo de resposta (JSON)
GET	/api/global	Retorna o estado completo da frota (Bateria, Posição, Status) e os últimos logs do sistema.	{"frota": [{"Rover-Alpha": {"bat": 98, "status": "IDLE"}}, "logs": [...]]}
GET	/api/telemetria	Versão leve do endpoint global, retornando apenas os dados de telemetria em tempo real para atualização rápida da interface.	{"Rover-Alpha": {"pos": [10, 20], "bat": 98}, ...}
GET	/api/rovers_lista	Retorna uma lista simples dos rovers registados para preencher menus de seleção.	[{"id": 1, "nome": "Rover-Alpha"}, ...]
POST	/api/enviar_missao	Envia um comando (Missão ou Carga) para um rover específico. A API converte este pedido HTTP num pacote UDP MissionLink.	Request: {"acao": "MISSAO", "target_id": 1, "missao": {...}} Response: {"status": "ok"}

Tabela 2: Tabela de endpoints implementados

Testes Realizados

Cenário e Configuração de Testes

Os testes visaram a validação da resiliência do protocolo aplicacional sobre UDP num ambiente de rede adverso simulado no CORE. A topologia envolveu uma transmissão simultânea para três rovers sujeitos a diferentes perturbações de rede apresentados abaixo:

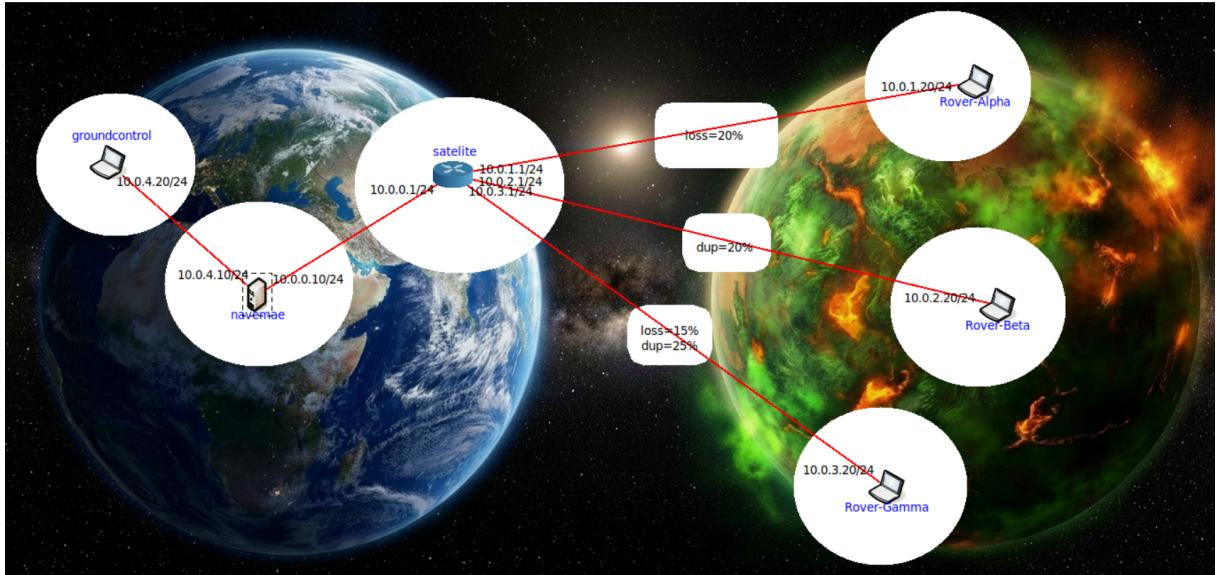


Figura 8: Tipologia para os testes realizados

- Rover-Alpha: 20% de perda de pacotes.
- Rover-Beta: 20% de duplicação de pacotes.
- Rover-Gamma: Cenário misto com 15% de perda e 25% de duplicação.

Análise dos Resultados

(Logs) Validação de Retransmissão (ARQ):

• Rover-Alpha:

```
[Rover-Alpha] ONLINE | Pos: [241, 328, 0]
[Rover-Alpha] A tentar estabelecer conexao com a Nave...
[Rover-Alpha] [ENVIO] 'STATUS: DESCONECTADO' (Seq 101) -> A aguardar ACK...[Rover-Alpha] A escuta (UDP)...
[Rover-Alpha] [TIMEOUT] Sem ACK (Seq 101). (1/5)...
[Rover-Alpha] [TIMEOUT] Sem ACK (Seq 101). (2/5)...
[Rover-Alpha] [ACK-RX] Sucesso! Nave confirmou Seq 101.
[Rover-Alpha] Conexao Estabelecida!
[Rover-Alpha] [ENVIO] 'STATUS: IDLE' (Seq 102) -> A aguardar ACK...
[Rover-Alpha] [ACK-RX] Sucesso! Nave confirmou Seq 102.
[Rover-Alpha] Missao Recebida: Fotografia de Alta Resolucao [ID: M-002]
[Rover-Alpha] [ENVIO] 'STARTED: Fotografia de Alta Resolucao' (Seq 103) -> A aguardar ACK...
[Rover-Alpha] [ACK-RX] Sucesso! Nave confirmou Seq 103.

[Rover-Alpha] [FOTO] A enviar dados binarios da Missao M-002...
[Rover-Alpha] [FRAG] Envio Seq 111 | Offset 0 | MORE_FRAGS (1)
[Rover-Alpha] [ACK-RX] Sucesso! Nave confirmou Seq 111.
[Rover-Alpha] [FRAG] Envio Seq 112 | Offset 250 | MORE_FRAGS (1)
[Rover-Alpha] [TIMEOUT] Sem ACK (Seq 112). (1/5)...
[Rover-Alpha] [ACK-RX] Sucesso! Nave confirmou Seq 112.
[Rover-Alpha] [FRAG] Envio Seq 113 | Offset 500 | MORE_FRAGS (0)
[Rover-Alpha] [TIMEOUT] Sem ACK (Seq 113). (1/5)...
[Rover-Alpha] [ACK-RX] Sucesso! Nave confirmou Seq 113.
[Rover-Alpha] [FIM] Foto enviada com sucesso.

[Rover-Alpha] [ENVIO] 'COMPLETED: Fotografia de Alta Resolucao' (Seq 119) -> A aguardar ACK...
[Rover-Alpha] [ACK-RX] Sucesso! Nave confirmou Seq 119.
```

Figura 9: Teste do Rover-Alpha

A imagem do log do Rover-Alpha demonstra a eficácia do mecanismo de fiabilidade implementado. Observam-se múltiplos eventos [TIMEOUT] Sem ACK (ex: Seq 101, Seq 112), causados pela perda de 20% configurada na topologia. O protocolo reagiu corretamente retransmitindo os pacotes até à receção do [ACK-RX], garantindo a entrega da missão sem corrupção de dados. Integridade e Descarte de Duplicados.

- **Rover-Beta:**

```
[Rover-Beta] ONLINE | Pos: [182, 212, 0]
[Rover-Beta] A tentar estabelecer conexao com a Nave...
[Rover-Beta] A escuta (UDP)...[Rover-Beta] [ENVIO] 'STATUS: DESCONECTADO' (Seq 101) -> A aguardar ACK...

[Rover-Beta] [ACK-RX] Sucesso! Nave confirmou Seq 101.
[Rover-Beta] Conexao Estabelecida!
[Rover-Beta] [ENVIO] 'STATUS: IDLE' (Seq 102) -> A aguardar ACK...
[Rover-Beta] [ACK-RX] Sucesso! Nave confirmou Seq 102.
[Rover-Beta] Missao Recebida: Fotografia de Alta Resolucao [ID: M-002]
[Rover-Beta] [ENVIO] 'STARTED: Fotografia de Alta Resolucao' (Seq 103) -> A aguardar ACK...
[Rover-Beta] [ACK-RX] Sucesso! Nave confirmou Seq 103.

[Rover-Beta] [FOTO] A enviar dados binarios da Missao M-002...
[Rover-Beta] [FRAG] Envio Seq 111 | Offset 0 | MORE_FRAGS (1)
[Rover-Beta] [TIMEOUT] Sem ACK (Seq 111). (1/5)...
[Rover-Beta] [ACK-RX] Sucesso! Nave confirmou Seq 111.
[Rover-Beta] [FRAG] Envio Seq 112 | Offset 250 | MORE_FRAGS (1)
[Rover-Beta] [ACK-RX] Sucesso! Nave confirmou Seq 112.
[Rover-Beta] [FRAG] Envio Seq 113 | Offset 500 | MORE_FRAGS (0)
[Rover-Beta] [ACK-RX] Sucesso! Nave confirmou Seq 113.
[Rover-Beta] [FIM] Foto enviada com sucesso.

[Rover-Beta] [ENVIO] 'COMPLETED: Fotografia de Alta Resolucao' (Seq 119) -> A aguardar ACK...
[Rover-Beta] [ACK-RX] Sucesso! Nave confirmou Seq 119.
```

Figura 10: Teste do Rover-Beta

Sob 20% de duplicação, o log do Rover-Beta mostra uma progressão linear dos números de sequência (Seq 101 -> 119). O sucesso da transferência da “Fotografia de Alta Resolução” confirma que o protocolo tratou corretamente os pacotes duplicados (provavelmente através da verificação de IDs de sequência), evitando o processamento redundante da mesma carga útil. O timeout pontual no Seq 111 sugere uma perda ocasional ou atraso excessivo no ACK, tratado com sucesso pela retransmissão.

- **Rover-Gamma:**

```
[Rover-Gamma] ONLINE | Pos: [217, 147, 0]
[Rover-Gamma] A tentar estabelecer conexao com a Nave...
[Rover-Gamma] A escuta (UDP)...[Rover-Gamma] [ENVIO] 'STATUS: DESCONECTADO' (Seq 101) -> A aguardar ACK...

[Rover-Gamma] [ACK-RX] Sucesso! Nave confirmou Seq 101.
[Rover-Gamma] Conexao Estabelecida!
[Rover-Gamma] [ENVIO] 'STATUS: IDLE' (Seq 102) -> A aguardar ACK...
[Rover-Gamma] [ACK-RX] Sucesso! Nave confirmou Seq 102.
[Rover-Gamma] Missao Recebida: Fotografia de Alta Resolucao [ID: M-002]
[Rover-Gamma] [ENVIO] 'STARTED: Fotografia de Alta Resolucao' (Seq 103) -> A aguardar ACK...
[Rover-Gamma] [ACK-RX] Sucesso! Nave confirmou Seq 103.

[Rover-Gamma] [FOTO] A enviar dados binarios da Missao M-002...
[Rover-Gamma] [FRAG] Envio Seq 111 | Offset 0 | MORE_FRAGS (1)
[Rover-Gamma] [ACK-RX] Sucesso! Nave confirmou Seq 111.
[Rover-Gamma] [FRAG] Envio Seq 112 | Offset 250 | MORE_FRAGS (1)
[Rover-Gamma] [ACK-RX] Sucesso! Nave confirmou Seq 112.
[Rover-Gamma] [FRAG] Envio Seq 113 | Offset 500 | MORE_FRAGS (0)
[Rover-Gamma] [ACK-RX] Sucesso! Nave confirmou Seq 113.
[Rover-Gamma] [FIM] Foto enviada com sucesso.

[Rover-Gamma] [ENVIO] 'COMPLETED: Fotografia de Alta Resolucao' (Seq 119) -> A aguardar ACK...
[Rover-Gamma] [ACK-RX] Sucesso! Nave confirmou Seq 119.
```

Figura 11: Teste do Rover-Gamma

O Rover-Gamma, sujeito à combinação mais agressiva de falhas (perda e duplicação), completou a missão M-002 com sucesso. O log evidencia a fragmentação correta dos dados binários (Offsets

0, 250, 500) com as flags MORE_FRAGS ativas, validando a capacidade do protocolo de reconstruir mensagens grandes mesmo em canais ruidosos.

Os testes confirmam assim que a implementação do MissionLink cumpre os requisitos de fiabilidade sobre UDP. O sistema lidou autonomamente com perdas, fragmentação e ordenação, assegurando que a Nave-Mãe e os Rovers mantêm a integridade da missão crítica e da telemetria independentemente da degradação do link.

Conclusão e reflexão crítica

Este trabalho prático permitiu-nos montar uma infraestrutura completa de comunicação para uma missão espacial, ligando a Nave-Mãe, os **Rovers** e o **Ground Control**. No geral, o sistema funcionou como esperado: conseguimos trocar mensagens críticas e ver a telemetria em tempo real, mesmo quando o emulador CORE estava configurado para “estragar” a rede com perdas e duplicados.

Pontos Fortes:

- **Fiabilidade no UDP:** O protocolo **MissionLink** cumpriu o objetivo. Mesmo com pacotes perdidos, o mecanismo de Stop-and-Wait garantiu que as ordens chegavam sempre ao destino (eventualmente). A fragmentação também funcionou relativamente bem, o que foi um desafio.
- **Estabilidade no TCP:** A decisão de usar um buffer que espera pelo caráter n foi simples, mas salvou-nos de muitos erros. Impediu que o sistema tentasse ler JSONs incompletos quando a rede partia as mensagens.
- **Multitarefa:** A arquitetura de threads separadas na Nave-Mãe (uma para UDP, uma para TCP, uma para HTTP) impediu que o sistema bloqueasse. O servidor conseguiu falar com os 3 rovers e com o browser ao mesmo tempo sem “congelar”.

Limitações:

Apesar de funcional, a nossa solução tem limitações claras que identificámos durante os testes:

- **O Rover com Loop Infinito:** Descobrimos um problema na nossa lógica de retransmissão persistente. Se o Rover não receber o ACK, ele entra num ciclo infinito de tentativas, recusando-se a desistir enquanto detetar que a Nave-Mãe está online (via TCP). Isto faz com que o Rover fique constantemente à espera, “preso” naquele pacote para sempre, o que na prática bloqueia a execução de outras tarefas se a falha no UDP for permanente.
- **Ineficiência do Stop-and-Wait:** Embora fiável, este método é lento. O emissor fica parado à espera de um ACK para cada pacote. Num cenário real com alta latência (como Marte-Terra), isto tornaria a transferência de ficheiros grandes (como as fotos) impraticável.
- **Polling no Ground Control:** O dashboard atualiza-se pedindo dados à API a cada segundo. Isto gera tráfego de rede desnecessário e podia sobrecarregar a Nave-Mãe se existissem muitos clientes.
- **Timeout Fixo:** O tempo de espera por ACKs é fixo (3 segundos). Se a rede ficar lenta, geram-se retransmissões desnecessárias; se for rápida, perde-se tempo à espera. O ideal seria um cálculo dinâmico.

Melhorias Futuras:

Se continuássemos este projeto, as prioridades seriam:

- Substituir o Stop-and-Wait de forma a enviar vários pacotes sem esperar.
- Implementar um **limite máximo** de tentativas globais para evitar que o Rover fique preso infinitamente num loop de envio. Ou utilizar o tcp para verificar se a Nave-mãe está operativa e assim ele ficar em “hibernação”.
- **Eliminar** o polling constante do browser.

Assim, pensamos que o projeto foi um sucesso porque cumpriu os requisitos e resistiu aos testes no CORE. De qualquer forma, há sempre espaço para otimizar a eficiência e melhorar o trabalho feito.